

GONE IN 29 SECONDS:  
WORLD'S FIRST

API HONEYPOT



# INTRODUCTION

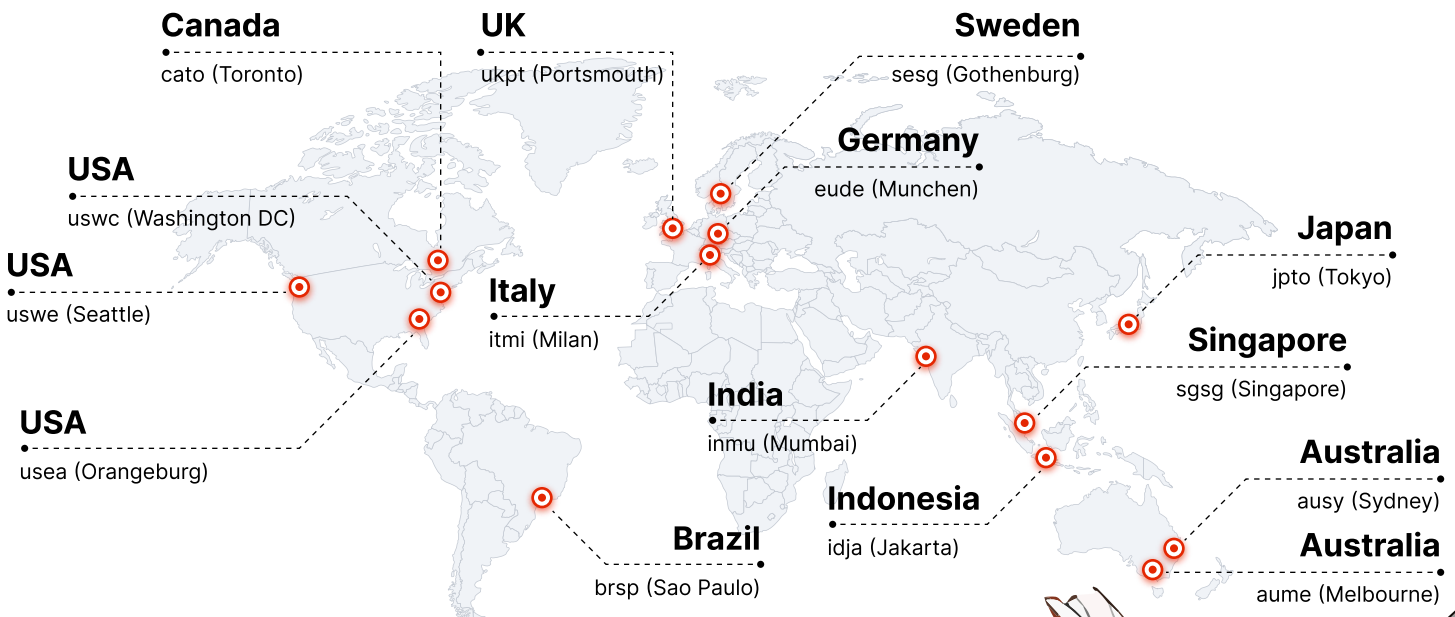
Honeypots are not a new concept. Distributed honeypots can provide valuable information about attackers and attack patterns, but they also serve as early warning systems in large enterprises. While there is general attack pattern data available from honeypots today, there is a gap when it comes to APIs.

In order to fill that gap, and to improve our understanding of the attackers and their tools, Wallarm deployed the first API honeypot in November of 2024. This report represents the initial results from 20 days of activity. We expected that it might take longer to have compelling data to report, but the speed at which our fake APIs were discovered and accessed surprised us. Instead of waiting for months of data, we decided to collect and report our immediate findings quickly. There's more work to be done, but the results so far are important, and worth sharing. Our initial objectives in creating this API honeypot architecture was to learn about attackers behaviour, patterns, and velocity. We were especially interested in determining a baseline time to discovery for newly deployed APIs.

The quality and speed with which we achieved our initial aims will no doubt drive us to make this report a recurring event.

## API Honeypot Architecture and Methodology

We had big ideas for our API honeypot architecture. We still do. But we have optimized our initial deployment for the objectives at hand, and for time to delivery. This research data has been collected from a mock API written in Golang with a self-signed SSL certificate. The API Honeypot is designed to log all the incoming requests, including storing full request bodies, for all possible 65535 ports and provide valid responses depending on the type of request, i.e. REST, GraphQL, etc. In our initial deployment, no domain names were assigned. All the honeypot instances were published on IP addresses distributed across 14 different locations around the globe:



*Ivan Novikov*

Ivan Novikov  
CEO, Wallarm



# TIME TO DISCOVERY



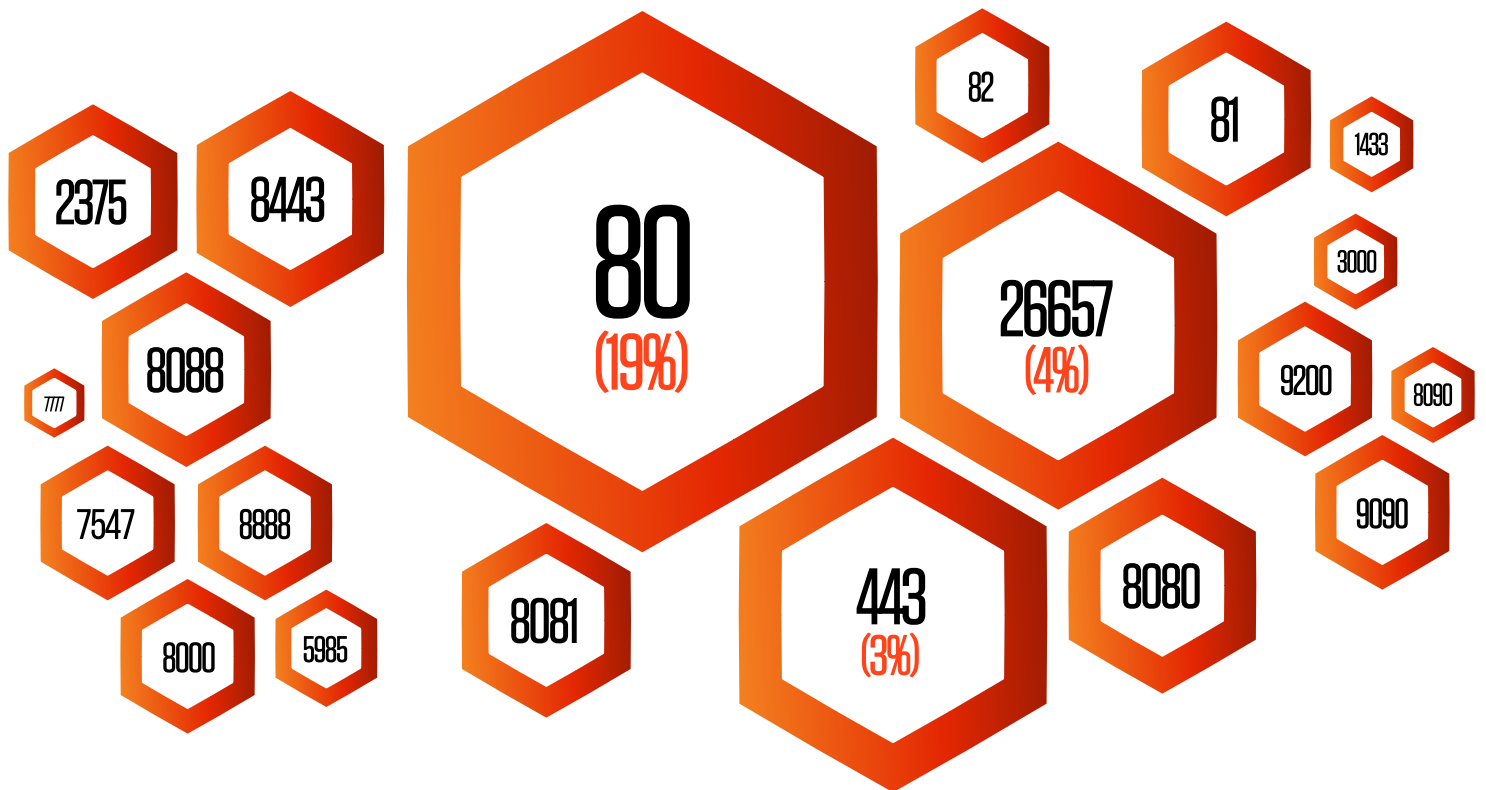
Our first objective was to understand how long it takes for a newly published API to be discovered, i.e. time to discovery. This metric is an important starting point, and benchmark, for API security. Newly deployed APIs are often less protected, unmanaged, and less secure. Our findings indicate that the average time for a newly deployed API to be discovered is just 29 seconds. Furthermore, the longest time we witnessed for discovery was 34 seconds. These two metrics are specifically the time from opening the port to the first API request to any of endpoints, excluding “/”.

Discovery of the port isn't the same as discovery of the API. We also found that the time to discovery for the API itself, in other words the time from port opening to a valid API call, was less than a minute.

# MOST COMMON PORTS

As noted previously, our API honeypot listens on all ports. This architecture is intended to capture information about which ports are commonly discovered and attacked. We expected that port 80 would be the most common, and we were not disappointed. Connections to port 80 accounted for 19% of the results. We were surprised, however, that the second most common port was not 443. It was actually 26657, followed by the usual suspects of 443, 8080, and 8443.

Other insights about the top 20 ports are less obvious. Port 2375 (docker REST API, unencrypted) came in at number 6. We also saw port 7547 (CWMP REST-based routers) in position number 11. This port was used by Mirai botnet back in 2016, and is still impactful for many older router models. Other ports of note are 5985 (WinRM HTTP-based API), 3000 (unPnP SOAP API, CISCO, Netgear and others), 1433 (commonly used for MSSQL, but now also used by many APIs as an alternative to default SSL 443 port), and 7777 (many different APIs).



APIs are not only 80, 443 and common known ports. Some very specific critical APIs are constantly under attackers' scans, like Docker's 2375 port, UnPnP SOAP, and CWMP HTTP-based APIs.



# MOST COMMON API ENDPOINTS

Second to the time to discovery metric, we had research objectives around understanding how attackers probe newly discovered APIs. While there's more work to be done here, including more complete emulation of APIs, the initial results are valuable. This table represents the ranking of API paths requested by clients when first connecting to the API honeypot.

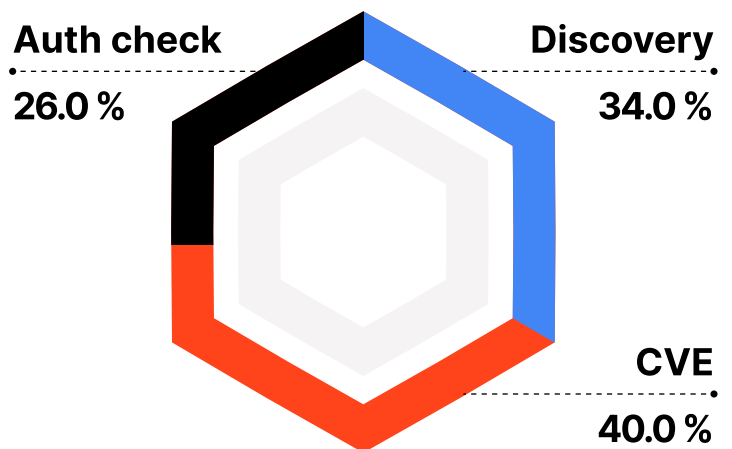
Rank	API Endpoint	Product/Service
1	/status	Generic
2	/v2/_catalog	Docker Registry
3	/manage/account/login	UniFi, Routers
4	/ws/v1/cluster/apps/new-application	Apache Hadoop YARN
5	/_ping	Docker, Generic
6	/v1.16/version	Docker
7	/config	Generic
8	/wsman	WS-Management
9	/version	Generic
10	/metrics	Prometheus
11	/bf/tracert	BigFix
12	/solr/admin/info/system	Apache Solr
13	/_cat/indices	Elasticsearch
14	/ipp	IPP Protocol
15	/query	Generic
16	/actuator/gateway/routes	Spring Boot Actuator
17	/solr/admin/cores	Apache Solr
18	/boaform/admin/formLogin	Tenda ONT GPON, Routers
19	/v1.24/containers/json	Docker
20	/geoserver/web/	GeoServer
21	/containers/json	Docker
22	/net_info	Generic
23	/api/sys/login	Generic
24	/remote/logincheck	Fortinet VPN
25	/all.json	Generic

From the data collected above, it's clear that you should not name your public and non-authenticated API endpoints with common names like /status, /info, /health or /metrics. We should note that the combination of port 26657 and the /status request points to Tendermint RPC, a blockchain component, as the probable service there. The prevalence of these paths in the data demonstrates that they are commonly probed. In other words, endpoints like these will be discovered in well under 2 minutes. If your service absolutely requires public, unauthenticated endpoints, it would be better to use less common names, or even better, use a random UUID or SHA256 hash, similar to the approach for webhooks.

We further analyzed the most commonly requested endpoints and mapped them to likely services and intent. Some of them were simply discovery calls to identify a product or service. Some were well-known product authentication checks, like Docker, Grafana, or Prometheus.

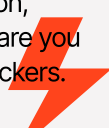
Some were attempts to exploit specific CVEs. We did an in-depth analysis for Top-50 the most common API requests observed to determine the type distribution:

## API Attacks by Types (among Top-50 most common API attacks)



Overall, our API honeypot registered about 337 unique API requests in the first few days after launching in any location, followed by a decreasing number of additional requests. We believe this behavior is the result of 403 responses returned during interactions, and plan to improve the capabilities with LLM-based responses in the future.

Check your Internet-facing APIs for authentication, before attackers do it. The more common software you use - the faster its API will be discovered by attackers.

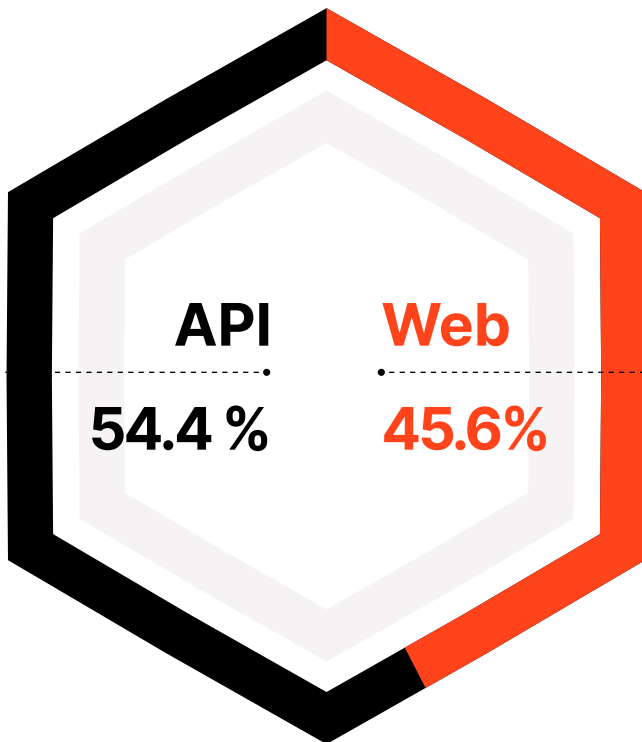


# WEB APPLICATIONS VS. APIS: WHICH ARE THE TARGETS?

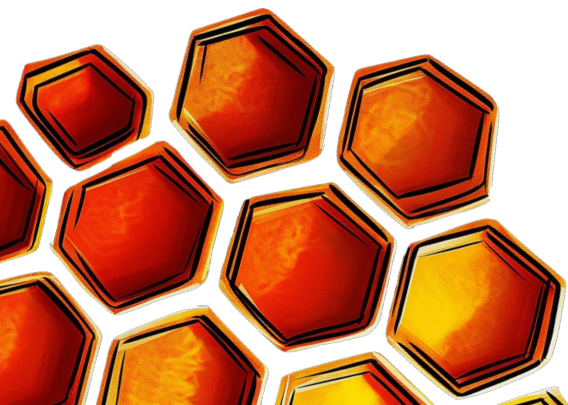
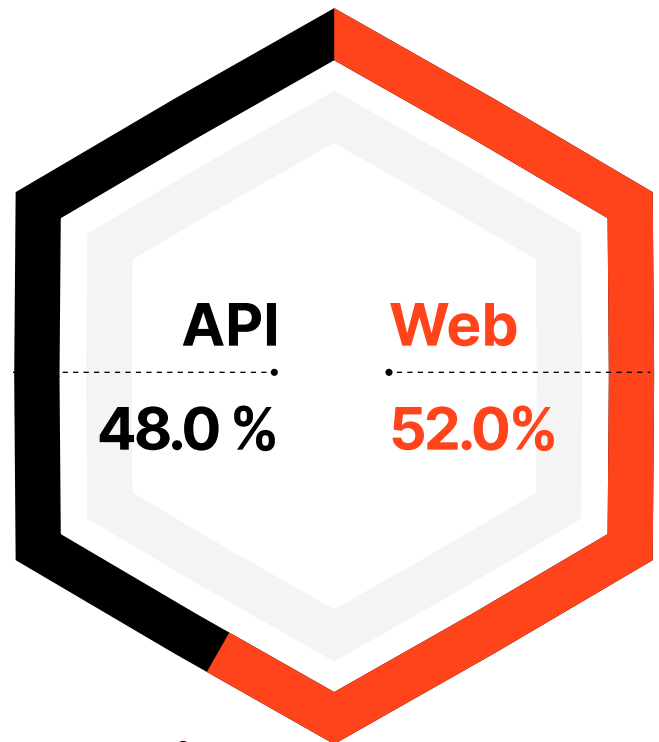
The honeypot observed that 54.4% of the total requests targeted APIs vs web applications, indicating that APIs have become a slightly more attractive attack surface compared to web applications, which accounted for 45.6% of requests. However, in terms of the diversity of unique exploits, web exploits made up 52% of the total, slightly outpacing APIs at 48%. We can compare these results to Wallarm's 2024 API ThreatStats report, which showed that 70% of attacks targeted APIs vs. web applications. The results are different because the data sets being analyzed differ, but the conclusion is the same: APIs are targeted more than web applications. This is particularly interesting because APIs only emerged as significant attack targets in recent years, whereas web applications have been a focal point for attackers for decades. This shift underscores how quickly APIs have risen in prominence within the threat landscape, demanding more attention from security teams despite their relatively recent entry into the spotlight.

<sup>1</sup><https://www.wallarm.com/resources/2024-api-threatstats-tm-report>

## Request Types



## Exploit Targets



# SOURCES: WHERE ARE ATTACKERS COMING FROM?

The analysis of ISP usage in API attacks highlights the strategic nature of these threats, as API attacks are not primarily volumetric, but instead focus on diverse and targeted exploit techniques. To better understand attacker behavior, we focused on unique exploits rather than sheer volume. Among the ISPs observed, Pfcloud UG, operating in Germany and The Netherlands, stood out with the highest diversity of unique exploits (748), reflecting a sophisticated and varied approach to API exploitation. Similarly, DigitalOcean, active across multiple countries, showcased significant diversity in exploit methods (743), underscoring how attackers leverage cloud platforms for flexibility and global reach.

Regionally focused ISPs such as China Unicom Beijing Province Network and Hydra Communications Ltd further emphasize how attackers leverage localized networks for unique exploit strategies. Additionally, ISPs like Zenlayer Inc and Alibaba showcase a multi-regional approach, with exploit activity spanning continents, from Asia to Europe. These patterns reveal a dynamic threat landscape where attackers employ both widely recognized and less conspicuous ISPs to maximize their reach and evade detection, highlighting the importance of monitoring exploit diversity across network sources.

Rank	isp	unique_explo	Countries
1	Pfcloud UG	748	
3	HOSTGLOBAL.PLUS LTD	743	
2	DigitalOcean	187	
4	AWS	149	
5	Azure	145	
6	China Unicom Beijing Province Network	101	
7	Linode/Akamai	95	
8	Hydra Communications Ltd	84	
9	Private Layer INC	77	
10	Zenlayer Inc	70	
11	CDS Global Cloud Co., Ltd	68	
12	DIGITALOCEAN	64	
13	Alibaba	60	
14	Shenzhen Tencent Computer Systems Company Limited	59	
15	GCP	55	
16	Alibaba	54	
17	Hangzhou Alibaba Advertising Co., Ltd.	52	
18	China Mobile communications corporation	50	
19	China Mobile Communications Corporation	49	
20	Oracle Cloud	54	
21	JSC Selectel	48	
22	China Unicom China169 Network	44	
23	UAB "Baltnetos komunikacijos"	44	
24	BITNET	44	
25	CAT Telecom Public Company Limited	44	

Attackers prefer to choose local providers to improve connectivity to your APIs to get more data faster and bypass geographic restrictions.















# HOW FAST ATTACKERS CAN STEAL API DATA?





During the API honeypot experiment, we observed API abuse patterns characterized by attackers using an average of **50 requests per second (RPS) distributed across 50 IP addresses**. Our estimates put this setup, achievable with minimal cloud infrastructure, at **\$50–\$150/month per IP**. This price point demonstrates the economic feasibility of large-scale API exploitation. Additionally, the bandwidth required for such attacks is minimal, making them stealthy and harder to detect in comparison to typical Distributed Denial of Service (DDoS) attacks.

We have analyzed how attackers exploit APIs through both single-request and batching techniques, and compared them with web scraping. In the appendix, you can also find technical details on batching attacks using XML-RPC and GraphQL.

## How fast attackers can steal 10 million user records

Attack Type	Avg Cost (50 IPs)	Bandwidth Usage (Mbps)	Time to Steal 10M Users (Minutes)
 <p><b>API</b> Single-request</p>	 <p><b>\$50–\$150</b> month</p>	 <p><b>~20 Mbps</b></p>	 <p><b>66 s</b></p>
 <p><b>API</b> Batching</p>	 <p><b>\$50–\$150</b> month</p>	 <p><b>~20 Mbps</b></p>	 <p><b>6 s</b></p>
 <p><b>WEB</b> Scraping</p>	 <p><b>\$10–\$50</b> month</p>	 <p><b>~2 Mbps</b></p>	 <p><b>1,666 s</b></p>

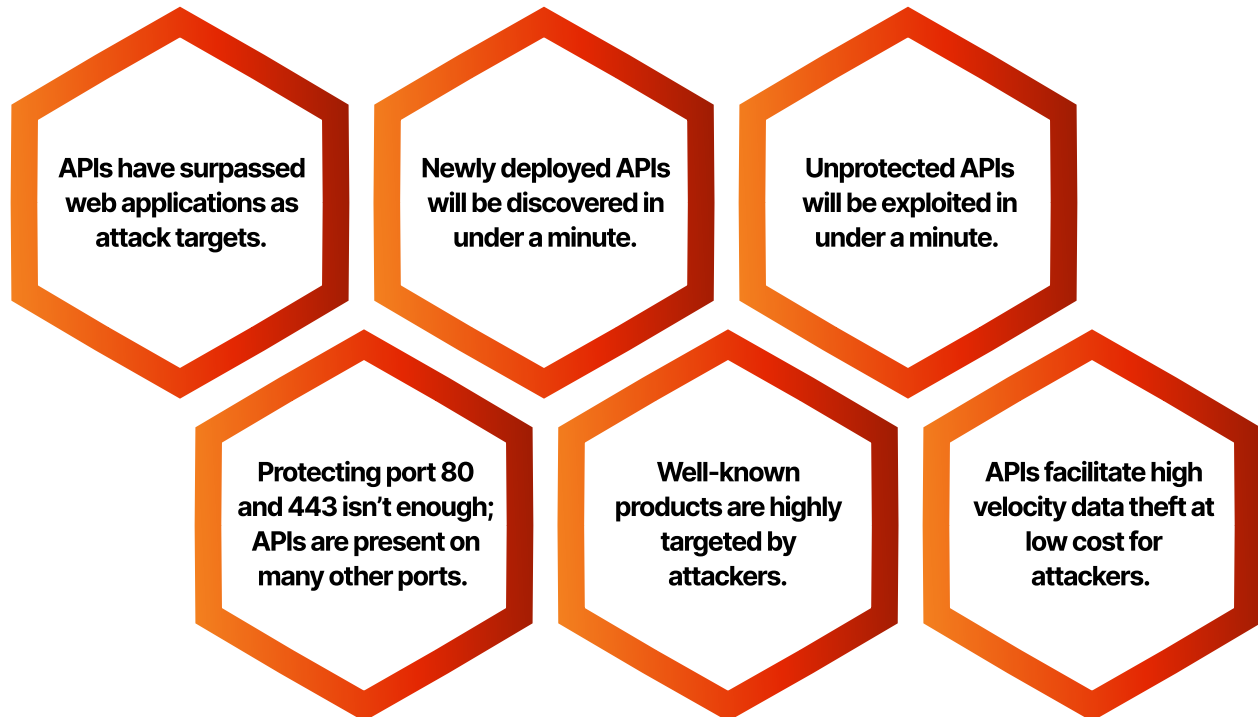
## Key Conclusions on API Attack Velocity

 <p><b>Cost-Effective Attacks</b></p> <p>With cloud infrastructure costs as low as \$50–\$150/month per IP, attackers can sustain abuse patterns like 50 RPS across 50 IPs with minimal investment.</p>	 <p><b>Stealthy Bandwidth Usage</b></p> <p>Even at peak efficiency, API attacks consume only ~20 Mbps—significantly lower than DDoS attacks, making them harder to detect through bandwidth monitoring.</p>	 <p><b>Batching Multipliers</b></p> <p>Techniques like XML-RPC and GraphQL batching can reduce time to steal 10 million records from over an hour to just minutes, highlighting the amplified risk.</p>	 <p><b>Web Scraping vs. APIs</b></p> <p>Traditional web scraping, though still a threat, is slower and more bandwidth-intensive, offering defenders more time to respond.</p>
--	--	--	--



# CONCLUSION

There is no dispute that the API attack surface is growing. API adoption is fueling business growth, and attackers follow the money. The API Honeypot report is intended to quantify specific metrics through empirical observation and analysis. Based on this data, here's what we know:



The conclusions should drive organizations to adapt existing security practices and adopt new security tools. Discovery of your API Attack Surface is a must, and protection from API attacks in real-time is a hard requirement. Wallarm will continue to produce valuable research to help organizations defend their APIs.

# APPENDIX

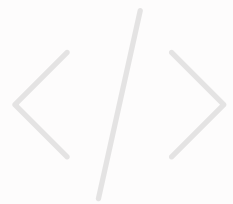
## What Are Batching Attacks?

Batching attacks exploit API features allowing multiple queries or operations to be combined into a single request. While designed for efficiency, this functionality can be abused to amplify the impact of a single API call. Two common implementations are XML-RPC and GraphQL.

### Example 1: XML-RPC Batching

XML-RPC APIs often allow batching multiple calls into one request. Below is an example of a malicious batch call designed to retrieve multiple user records:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>system.multicall</methodName>
  <params>
    <param>
      <array>
        <data>
          <value>
            <struct>
              <member>
                <name>methodName</name>
                <value><string>getUserDetails</string></value>
              </member>
              <member>
                <name>params</name>
                <value><array><data>
                  <value><int>1</int></value>
                </data></array></value>
              </member>
            </struct>
          </value>
          <value>
            <struct>
              <member>
                <name>methodName</name>
                <value><string>getUserDetails</string></value>
              </member>
              <member>
                <name>params</name>
                <value><array><data>
                  <value><int>2</int></value>
                </data></array></value>
              </member>
            </struct>
          </value>
        </data>
      </array>
    </param>
  </params>
</methodCall>
```

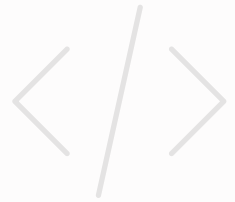


Here, the attacker requests user details for multiple IDs in a single call, reducing the number of requests needed and evading rate limits.

## Example 2: GraphQL Batching

GraphQL APIs allow batching through its query language, which attackers can use to request multiple resources simultaneously. Below is an example query:

```
query {  
  user1: getUserDetails(id: 1) {  
    id  
    name  
    email  
  }  
  user2: getUserDetails(id: 2) {  
    id  
    name  
    email  
  }  
  user3: getUserDetails(id: 3) {  
    id  
    name  
    email  
  }  
  # Up to 100+ user queries combined  
}
```



This query retrieves details for multiple users in a single request, exploiting the GraphQL batching capability to exponentially increase data retrieval speed.